APPENDIX II

(Marked-Up Copy of Original Specification as Required by 37 C.F.R. §1.125(b))

ARCHITECTURE FOR INTELLIGENT AGENTS AND A DISTRIBUTED PLATFORM THEREFOR

Field of the Invention

This invention relates to the field of artificial intelligence, and, in particular, to software objects commonly known as intelligent agents.

Background of the Invention

~~Semantic Web~~

The web is fast evolving from being a repository of information for human consumption - the syntactic web - to something that carries much richer forms of information - the semantic web.

The current generation of the web is geared towards presenting information to human beings. The lingua franca of the web - HTML - essentially describes how the information in a web page is to be presented in a browser to a human being. The web has, of course, come a long way in the past few years. It started out serving just static content. Then it incorporated dynamic content generated from databases, and it has now moved on to also become a front-end to complex n-tier transaction systems. But all the interesting, powerful interactions between computer systems in the world today happen in islands of information. As soon as information passes through a web front-end in the form of HTML, the ability of non-humans to make use of this information goes down drastically.

The vision of the Semantic Web has been espoused to get around precisely this bottleneck. As the amount of information present on the web grows ~~and grows and grows~~, it becomes very important for this information to be in a form that programs can understand and process. That is where XML, the language of choice for the semantic web, comes in. XML allows us to mark-up information with syntax that is meaningful in a specific domain. When this

domain-specific syntax is combined with associated semantics specified by an ontology[1], we have all of the prerequisites that are necessary for programs to really understand and make full use of the information on the web - providing tremendous value to consumers and businesses, and to individuals and organizations.

Agents

Once the basic infrastructure for the semantic web is in place, we have everything that is required for the existence of a class of very powerful programs called agents. The word agent is a heavily over-used one, and over the years many definitions have been given for the term. As far as we are concerned, an agent is a computer program with the following characteristics:

- Autonomy: Agents operate without the direct intervention of humans.
- Reactivity: Agents respond to events in the environment in an appropriate manner.
- Goal Directed Behavior (Pro-activity): In addition to reactivity, where they respond to changes in the environment, Agents also exhibit goal directed behavior by figuring out for themselves that they need to do certain things to attain user goals.
- Social Ability: Agents communicate with other agents at a high level of abstraction, using an agent communication language, to get work done.

Context Aware Computing

Context aware Aapplications are application that of this kind derive their name from the fact that they keeps track of the current context of an entity like such as a human user – specified by features like location, identity, time, activity, events – and then provide services that make the most sense in this context.

Context aware services can broadly be broken down into the following categories:

- Automatic triggering of actions depending on context: e.g. a tour guide that informs a user of nearby places of interest based on the user's profile.

---

[1] An ontology is the specification of a model: the core entities of interest in a domain, the relationships between these entities, and the various things of interest that can be done with these entities.

- Presentation of information to the user on request: e.g. telling the user where the nearest coffee shop or nearest bank is.

~~This patent discloses an infrastructural technology, in the form of a generic agent architecture, upon which domain-specific intelligent agents can be built, and a distributed artificial intelligence platform upon which the intelligent agents can be deployed. One of the areas where this agent technology can be applied in a very powerful way is context aware computing.~~

Summary of the Invention

Disclosed herein is an infrastructural technology, in the form of a generic agent architecture, upon which domain-specific intelligent agents can be built, and a distributed artificial intelligence platform upon which the intelligent agents can be deployed. One of the areas where this agent technology can be applied in a very powerful way is context aware computing.

The distributed artificial intelligence platform disclosed herein includes a very scalable distributed architecture capable of simultaneously running millions of context monitoring agents. The architecture has built-in support for fault tolerance in the face of software and hardware failures, and allows for the transparent resurrection of agents on different hosts if the server they are running on goes down, in a manner that is guaranteed to preserve semantic correctness.

Also included a general-purpose layered agent architectural framework, which includes core technology for reasoning, planning, optimization, constraint satisfaction, communication, workflow management, and learning. Using this framework, domain-specific technology can be easily embedded inside different kinds of agents. Equipped with the abilities provided by these core technology building-blocks, agents deployed on the platform will have the ability to define and fully participate in the next generation of intelligent distributed systems running on the semantic web.

Detailed Description of the Drawings

Figure 1 shows the layered architecture of an intelligent agent according to this invention.

Figure 2 shows a detailed architectural diagram of an agent, showing the movement of events and messages.

Figure 3 shows the components of a distributed platform to support intelligent agents.

Figure 4 shows the flow of data within the distributed platform.

Detailed Description of the Invention

~~The word agent is a heavily over-used one, and over the years many definitions have been given for the term. For purposes of this disclosure, we consider an agent to be a computer program having the characteristics of autonomy, reactivity, proactivity (goal-directed behavior) and social ability (communications with humans and other agents).~~

The basic distributed platform disclosed herein is modeled after the traditional notion of an operating system, which~~,~~ provides, amongst other things, an interface over bare hardware. At the end user level, an operating system provides a friendly set of applications that make it possible for a user to productively use a complex piece of hardware without ~~without knowing a whole lot about it~~knowledge of the details of its design or internal workings. At the programmer level, it provides a system call interface that makes it possible for a programmer to program complex operations without bothering about low level hardware details and without worrying about conflicts with other programs.

The distributed platform provides a similar interface over a network capable of deploying ~~A~~agents. This could be a local area network, or it could be the whole ~~w~~World-~~w~~Wide ~~w~~Web in its next incarnation as the semantic ~~W~~web. This interface, too, works at two levels. At the end user level, it provides a set of intelligent web~~-~~based applications in different domains powered by intelligent agents. At the developer level, it provides a set of re-useable, pre-built agents and components that can be used to easily build intelligent web applications ~~applications without a lot of effort, and~~ without ~~worrying too much~~concern regarding ~~about~~ communications and artificial intelligence issues.

Agent Architecture

For an agent to exhibit the characteristics that we identified earlier, it needs to have certain essential capabilities. These include the ability to reason, the ability to plan, the ability to learn from its actions and the ability to communicate with other agents and information systems. The architecture disclosed herein enables the agents to have these characteristics.

Figure 1 shows the layered architecture of a generic agent in the preferred embodiment. The architecture of an agent is complex, and an agent is capable of doing a variety of different tasks, at several levels of abstraction. For this purpose, an agent is decomposed into layers, so that different layers can specialize ~~in doing their own thing, and can~~ collaborate with other layers to provide powerful overall functionality.

The various layers of the agent architecture communicate with each other via events, which are published on event bus 60. Any layer of the architecture can publish events on bus 60 or read events from bus 60 and act upon them. When a layer receives some input in the form of an event, it processes it, and publishes the result of the processing as a new event on bus 60. Any layer interested in this new event is welcome to pick it up and do its own processing ~~on the~~ in response to the event. This interlayer, event-based communication between layers is asynchronous, and is designed to be capable of very high event throughput.

The collaboration layer 10 is responsible for handling incoming and outgoing messages. With respect to outgoing messages, collaboration layer 10 verifies and directs outgoing messages to other agents. With respect to incoming messages, collaboration layer 10 determines whether the agent is interested in the incoming message. Each message has a field that identifies the language in which the message is encoded, and another field that identifies the ontology that the message conforms to. ~~The c~~Collaboration layer 10 contains an interpreter-table 15. This table has entries for at least one interpreter for each domain-specific language and ontology that the agent supports. As a result of the interpretation of the message, collaboration layer 10 may publish an event on event bus 60. As an example, with reference to Figure 2, agent 5 receives message 12 containing a new rule that agent 5 needs to run. Collaboration layer 10 receives message 12 and looks up an interpreter from the interpreter-table 15 that is capable of handling the message. It then hands the message to ~~this~~the interpreter for decoding. The interpreter has an ontology handler that decodes the message and publishes a "New-Rule" event 14 on event bus 60.

Reasoning layer 30 and belief layer 40 are part of an expert system that is the core of the agent architecture. Although conceptually, reasoning and belief functions can be thought of as separate, in reality, they may be implemented in a powerful way as an expert system 35. As a result, we will often refer to them herein as "reasoning/belief layer 30/40". Belief layer 40 comprises a collection of facts 42 in the working memory of expert system 35. Reasoning layer 30 comprises a database of rules 32 in the knowledge base of expert system 35 and the execution of the rules within the inference engine of expert system 35. In our example with the "New-Rule " event 14, reasoning layer 30 would read event 14 from event bus 60 and place the new rule contained in event 14 into the rule database 32 of expert system 35. Reasoning layer 30 also contains logic engine 34, which helps in the evaluation of rules 32 based on the current set of facts 42 in the working memory of expert system 35.

Sensory layer 50 is responsible for monitoring the world outside of agent 5 and reporting changes in the environment to the other layers of the agent 5. To this end, sensory layer 50 will monitor sensors 52 and gather regular sensor updates. When a change is sensed, sensory layer 50 will publish a "New-Fact" event 16 on event bus 60.

Reasoning / belief layer 30/40 reads "New-Fact" event 16 from event bus 60 and will update fact database 42 with the new fact. This causes an evaluation of the rules with rules database 32 of expert system 35 that might fire because of the presence of this new fact. As part of this evaluation, calls are potentially made into logic engine 34 to check to see whether certain conditions hold. This feature gives each agent access to not only the powerful and expressive logic-programming paradigm, but also to additional features such as constraint satisfaction and optimization that are present in the logic engine. This allows reasoning layer 30 of agent 5 to be as complex and powerful as required for the particular function to be carried out by agent 5. ~~And this can all be configured at run time, without any overhead for a simple agent that doesn't need all these features.~~

The action layer 20 of agent 5 is responsible for taking in pending actions and storing and carrying out actions required by agent 5. For example, the evaluation of the rules in rules database 32-, based on facts stored in facts database 42, may require that agent 5 undertake a certain action. In this case, an "Action-Event" 18 is published on event bus 60 by reasoning/belief layer 30/40. Event 18 is picked up by action layer 20 from event bus 60, and the appropriate action 19 is carried out.

Logic engine 34 is the technology that provides core reasoning ability to an agent.

Logic engine 34 provides support for a style of programming known as logic programming. Logic programming is based on ideas derived from the discipline of formal knowledge representation and reasoning. Formal logic is used to represent knowledge about specific domains, captured in the ontology for each specific domain, and to reason about things in these domains. It consists of the following:

- A basic alphabet of symbols, which stand for things of interest in a domain.
- Valid syntax, defined by a grammar, which describes how to make sentences from these symbols. These sentences state facts about the domain.
- Semantics, which provides an interpretation for sentences i.e. how sentences relate to state of affairs in the domain.
- Proof theory - made up of rules of inference for deducing new sentences from old ones.
- A set of axioms that describe the known set of facts and rules about a domain.

A logic programming language is an embodiment of these ideas. It provides syntax and a proof theory for representing knowledge about a domain. The programmer specifies axioms describing the domain, and provides an interpretation for statement. Queries posed to the system are answered on the basis of the axioms by the application of the rules of inference of the system. Contrast this with the imperative programming, where the programmer specifies exactly what to do and when to do it.

Preferably, ~~L~~logic engine 34 is implemented as an interpreter for a dialect of Prolog that has been extended to give it the ability to reason about numbers and constraints/optimization-problems based on numbers. Most ~~P~~preferably, the logic engine is a pure Java implementation of a superset of Prolog. It features very good interoperability between the logic programming model and the well-known object oriented programming model, thereby allowing access to a wide range of Java application programming interfaces (APIs) from a logic program. This interoperability, for example, enables the access of multiple databases ~~(through JDBC)~~ in a single logic query. When implemented in the preferred manner, ~~T~~the logic engine has a very

extensible architecture that allows the plugging-in of predicates written in Java. ~~In fact, all the Prolog built-in predicates within the Logic Engine have been implemented as independent Java classes.~~

The logic engine is the key component within an agent that provides support for information extracted from the semantic web. This symbolic information, marked-up in XML or a dialect of XML know as RDF, can be imported into the logic engine. From that point onwards, this information can be used as a basis for 'reasoning' - in situations that require use of this information.

The logic engine also features a constraint satisfaction ability. This facilitates the efficient solving of a large class of numeric problems that would otherwise be difficult to solve based on just the Prolog programming model. Some of features of this subsystem may include:

- Node, arc and bounds consistency based solvers for Finite Domain Constraints
- A solver based on Integer Programming (branch and sound) for Finite-Domain Optimization problems
- Support for Real-Domain constraints/optimization based on the Simplex method.

An important aspect of the distributed platform is the ability of agents to learn the best way to complete a given task. ~~In our view,~~ ~~a~~An agent is said to learn how to do a task based on the experience of doing that task, if some performance measure associated with the agent's doing the task improves as the agent gains more experience.

In computational terms, it is very important to decide on the exact nature of the knowledge that is to be learned by the agent. In general, it is useful to define an entity called the target function to represent the knowledge to be learned. The target function should be such that the process of learning the function leads to improved performance by the agent on the task under consideration. Given this conceptual framework, a machine learning algorithm can be said to consist of a target function that is learned and a way of learning this function from training data or experience

The framework disclosed herein contains implementations of the following:

- **Decision Trees**, which can be used to learn discreet valued functions. We disclose a special kind of decision tree called an identification tree;

- **Neural Networks**, which can be used to learn discreet valued or real valued functions; and

- **Reinforcement Learning**, which is used for learning optimal control policies for an agent in an uncertain and non-deterministic environment.

What follows is a brief description of each of these algorithms.

Decision trees are used to classify data. The discreet valued function that is learned is a mapping from a set of attributes that characterize an instance of data to a category. A decision tree classifies an instance by sorting it down the tree from the root to a leaf node. Each node in the tree corresponds to a test on an attribute of the instance, and each branch descending from the node corresponds to one of the possible values of the attribute. The leaf nodes correspond to the different categories that define the possible classifications.

Once a decision tree is built, it is straightforward to convert it into a set of equivalent rules. We get a rule by tracing the path starting from a leaf node ending at the root node. By repeating this for all the leaf nodes a set of rules is obtained. After devising a rule set, useless rules must be eliminated. The question to be asked is: can any of the test outcomes be eliminated without changing what the rule does to the sample. If yes, it should be eliminated, thereby simplifying the rule set.

Preferably, the decision tree will have some means of reading training data from a relational database management system and a means of saving a decision tree in to a database.

Neural networks provide a powerful solution to the problems of both function approximation and classification. They have been used in a wide variety of tasks, ranging from character recognition to reinforcement learning action-value function approximation.

The implementation of the neural network for the distributed platform is preferably of the Multi-Layer-Perception type. This type of neural network has more than one layer of adaptable weights, and can be used to learn any kind of non-linear function.

The most important part in training a neural net is the weight update method, called back propagation. It helps search the most optimal configuration of the neural net. Preferably, the neural network component of the machine learning subsystem will have support for weight

decay, support for use of momentum and support for variable search rates (e.g. RProp where the search rate automatically reduces if we are far from our goal and vice versa).

Reinforcement learning is concerned with the issue of how an agent, situated in an environment that it senses, and on which it acts, can learn an optimal control policy to achieve its goals. Or to be more precise, how an agent can learn to map situations to actions - so as to maximize a numerical reward signal. The agent is not told which actions to take, but instead must discover which actions yield the most reward by trying them. The target function that is learned here is something called the action-value function, and it assigns a numerical value to each action available in a state. Learning an optimal control policy boils down to learning the optimal action-value function.

In reinforcement learning, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of reinforcement learning.

Reinforcement learning is defined not by characterizing learning algorithms, but by characterizing a learning problem. Any algorithm that is well suited to solving that problem can be considered a reinforcement learning algorithm. The basic idea is simply to capture the most important aspects of the real problem facing a learning agent interacting with its environment to achieve a goal. Clearly, such an agent must be able to sense the state of the environment to some extent and must be able to take actions that affect that state. The agent must also have a goal or goals relating to the state of the environment.

The job of a reinforcement learning agent is to maximize, over a given period of time, the numerical reward signal that it receives from the environment. However, it is impossible to write a single agent that solves all reinforcement learning problems. There are different kinds of environments, for example, finite or infinite, episodic or non-episodic, deterministic or non-deterministic, stationary or non-stationary. Depending on the nature of the environment, the agent must choose an optimal type of value function. So, for example, agents with tabular value-functions can interact only with finite environments. Similarly different kinds of planning agents would be appropriate for deterministic and non-deterministic environments.

A reinforcement learning agent must learn which action is good and which is bad by trial and error. Thus, an agent needs to completely explore all possibilities before deciding which one

is the best. However, for optimal performance, it should always select the best action encountered so far. This inhibits explorations. A soft-max action selection criterion spares the agent this dilemma. However, there are many kinds of action selection methods, each of which may be suitable in a different kind of environment.

The most important aspect of a reinforcement learning agent is learning from the response of the environment (i.e. the reward signal). The agent compares the actual reward to the expected reward. Using these errors as a starting point, it learns and refines its policy using backup evaluation and credit assignment methods. Different agents may use different types of backup evaluation and credit assignment methods to achieve optimal performance.

Discussed herein are some aspects of the behavior of a reinforcement learning agent. After factoring in all of them, it should be obvious that hundreds of different agents may be needed for different scenarios. To overcome this problem we, have come up with a generic architecture that allows mixing and matching of capabilities. This enables custom-creation of an agent. Each aspect is characterized by a module, which can be implemented in several ways. Each one of the implementations may have their advantages and disadvantages. Moreover, different implementations may cater to different kind of problem. In a nutshell, this architecture supports a library of implementations of the various modules, which must be correctly chosen depending on the nature of the problem.

Typical modules include:

- Action selectors: Boltzmann, Epsilon
- Backup evaluators: TDSarsa, TDQ, TTD
- Credit assigners: Waltkins, Eligibility Traces
- Value Functions: Tabular, Linear, Non-Linear
- Planner: DynaQ, DynaQ+, DynaAC, Prioritized Sweep

One obvious advantage of this architecture is that we can create an agent suitable for our needs by assembling the appropriate modules. Practically, this may be as easy as just writing a configuration file or assembling things with a few mouse clicks. Another advantage is that this architecture is extensible. A new planning algorithm, for example, can be added to the library of

existing implementations and then can be used while assembling the agent without any modification to the rest of the reinforcement learning framework.

Distributed Platform Architecture

This invention also defines a distributed architecture for support of intelligent agents. Note that it is not necessary that the intelligent agent conform to the architecture above. An intelligent agent of any design can be supported by the distributed platform, as long as it is capable of processing and responding to the messages sent to it. For purposes of this disclosure, the distributed platform will be referred to as a "society." The architecture for the society is composed of several logical elements, which will be discussed below. The society is physically supported by many hardware elements, including a plurality of computers and a communications network, such as the ~~i~~Internet, to interconnect the computers together

The society is a scaleable distributed architecture capable of simultaneously running millions of context monitoring agents. The society has build-in support for fault tolerance in the face of software and hardware failure and allows for the transparent resurrection of agents on ~~different~~alternate hosts if the server that they are running on ~~goes down~~fails, such that ~~in a manner that is guaranteed to preserve~~ semantic correctness and context are preserved.

The society is capable of supporting three basic types of agents. These are entity agents, stateless task agents and stateful task agents. Entity agents generally represent entities that are interacting with the society, such as a ~~physical~~ human user. These agents are generally very long lived and do context aware computation within the society. Generally the entity agents always need to be running because they need to continuously monitor the context of their corresponding entity (eg. a human user) ~~of the society~~ for specific conditions. Entity agents are capable of surviving server crashes transparently to the client. One example of an entity agent would be a user agent that monitors a user's location in the physical world to inform him of things of interest that are nearby. Physical users who register with the society will have an entity agent constantly running on their behalf, awaiting instructions from the user regarding a task to be carried out.

Stateful task agents carry out specific, multi-step tasks for a client, which may require the recall of previous states or the results of previous steps as the steps of the task are executed. Thus, as state context is maintained between steps of the multi-step task, Stateful task agents

maintain a conversation session with a client that is interacting with them. This session typically expires when the task has been completed.

Stateless task agents are similar to stateful tasks agents except they do not retain a context and are generally useful for carrying out single step tasks, for example, the sending of an e-mail on behalf of a user. ~~These agents are generally short-lived and expire at the completion of the single-step task that they had been instructed to carry out. Thus, no context need be retained within a stateless task agent.~~

In Figure 3., agent hosts 110 and 120 in the society are processes on physical computers which house agents and manage their life cycles. Agent hosts also act as message dispatchers for all of the agents they are supporting. ~~living within them.~~ Agent host 110 represents a host capable of hosting entity agents, Agent host 120 represents a host capable of hosting stateful task agents, while Agent host 121 represents a host capable of hosting stateless task agents. These three kinds of hosts are different in structure because they specialize in controlling the different lifecycles of their respective agent types, and provideing services to these agents. Typically in a society, there may be dozens or hundreds of agent hosts. Typically, an agent host is a process on a single physical computer able to communicate with other elements of the distributed platform via a communications network such as the ~~i~~Internet.

Facilitator 160 facilitates communication within the society. ~~Facilitator 160~~and is the primary interface for communications between the entities within the society and the outside world., ~~first and foremost provides a facade to the society with~~ Facilitator 160 has multiple high level ~~helpful~~ methods to assist an entity outside the society in dealing with agents and services running within the society. Any outside entity, such as a user interface program, running on behalf of a user who has an agent deployed on his behalf within the society, can communicate to the society through the facilitator. Secondly, ~~the~~ facilitator 160 controls the agent activation protocol in the case of communications failure between a client of an agent and the agent host supporting ~~within which~~ the agent ~~is running~~. Additionally, ~~the~~ facilitator 160 houses servers for white pages 162, yellow pages 164 and agent activator 166. White pages server 162 provides information on specific agents and users, such as the address of the agent host where the agent is running, the agent's identifier, etc. Yellow page server 164 provides information about agents or sets of agents that provide services to the society. Agent activator 166 controls the agent activation protocol.

Workflow manager 150 controls the execution of multi-step tasks on behalf of the agents. ~~The w~~Preferably, workflow manager 150 is implemented as a stateful task agent that manages workflows within the system. The main components of the workflow subsystem are outlined below.

Within workflow manager 150 is process definition repository 152. It is a component of the workflow subsystem in which definitions of business processes or other kinds of multi-step processes are stored. These definitions are preferably in the form of XML documents. ~~The r~~Repository 152 is accessible over the network and allows different versions of the same process to be stored. In this way, changes can be made to ~~the~~ a process definition without effecting processes that might already be running.

Workflow engine154 is the heart of the workflow subsystem. It is responsible for running the workflows. It parses the process definitions, decides the tasks or task steps that are to be activated next, and sends out notifications when necessary.

In the preferred embodiment, b~~B~~ecause the workflow manager 150 is a stateful task agent, it is running under a stateful task agent host 120. This host is responsible for managing the workflow agent's life cycle and also acts as a dispatcher for incoming messages addressed to the workflow manager.

There are task agents participating in the workflows being managed by ~~the~~ workflow manager 150. A workflow may be triggered by an external event (for example, in response to a 10% change in the cost of an airline ticket). A request to trigger the workflow must contain any relevant contextual information and the information regarding specific agents who shall participate in the process. Agents may be contacted by the workflow engine on the basis of their names (a lookup in white pages server 162) or on the basis of services they perform (a lookup in yellow pages server 164). A third way to contact the agents is by means of their agent identifier, which corresponds to a low level address as opposed to a name/service category-based lookup.

A workflow may involve several tasks being farmed out to a number of agents in parallel. Each agent asynchronously notifies ~~the~~ workflow engine 154 when its assigned task has been completed. ~~The~~ Workflow engine 154 then performs a dependency analysis to figure out if the workflow can proceed further and what new tasks need to be assigned.

Database 130 is a central repository of persistent information in the society. This information includes such things as context rules for context aware agents and conversation

information for stateful task agents. LDAP directory 140 is the central naming service in the system that houses administrative information such as user ids and user descriptions, agent ids and agent descriptions and host ids and host network endpoint descriptions. Agents are activated within the society via an agent activation protocol.

All network messages for an agent are sent to an entity called the message dispatcher, shown in Figure 3 as 112 within each agent host 110, ~~or~~ 120 or 121. ~~The m~~Message dispatcher 112 in each agent host accepts messages for agents running on that host. Addresses for the agent's message dispatcher are looked up from the society LDAP directory 140. Message dispatchers 112 are implemented using a special kind of remote object that supports fault tolerance and load balancing capabilities. For this kind of remote object, stubs that are delivered to a client to enable remote calls are smart. If the stub detects a network failure that makes it impossible for the stub to communicate with its server-side message dispatcher, the stub, instead of returning an error to the client, instead forwards an agent activation request to agent activator 166 within facilitator 160. Facilitator 160, on receiving a request for agent activation for agent 5, for example, puts an activation request into a piece of distributed memory 170 called the collaboration space 170, that is shared amongst all entities in the society. Any one of the entity agent hosts 110 running in the society can pick up the activation request from the collaboration space 170 and activate the agent 5. Once agent 5 is up and running the end point information for the agent's dispatcher is sent back to the stub running within the client process. The stub gets updated to point to this new location and a call is made to this new message dispatcher to forward the client's message to its agent. All this happens in a manner that is totally transparent to the client, through the use of the smart stub.

Figure 4 illustrates the flow of messages within a society as a user client 200 interacts with its entity agent 5. Client 200 may be any one of a variety of well known means of interacting with the society, such as a dedicated software tool, an applet or a browser. In step 101, the client queries facilitator 160 for the location of white pages server 162. In step 102, client 200 queries -white pages server 162 through facilitator 160 to discover the location of agent 5. (i.e., what agent host the agent is running on). White pages server 162 will access LDAP directory 140 to get the latest information about agent 5. In step 103, client 200 sends a message to agent 5. The message is picked up by agent host 110 which is hosting agent 5. Message dispatcher 112 running in agent host 110 receives the message and routs it to agent 5.

In step 104, a sensor within agent 5 fires and queries a task agent 7 for some sensory information. Message dispatcher 112 in task agent host 120 receives the message for task agent 7 and routes it to task agent 7. Task agent 7 may use any means it chooses to process the request (internal computation, ~~proprietary~~proprietary network protocol to access an entity outside the society etc). In step 105, the information received from task agent 7 is placed into the belief layer of agent 5, and, causes a re-evaluation of the rules in the expert system of agent 5. As a result of the re-evaluation of the rules in the expert system of agent 5, a rule fires, and a workflow is initiated in step 106. In steps 107 and 108 , the workflow manager makes use of different stateful or stateless task agents to carry out the tasks in the workflow. This accomplishes whatever client 200 wanted to get done on the triggering of the context of interest.

As a real life example of the process flow shown in Figure 4, consider ~~that~~ a user ~~201~~ who needs to go to the post office. As a reminder, ~~H~~he wishes to have a message delivered to his cellular telephone the next time he passes within five hundred yards of a post office ~~as a reminder~~. To do this, the user ~~201~~ must use client 200 to author a rule to be sent to his entity agent 5. Once the user ~~201~~ has authored the rule, facilitator 160 is contacted by client 200 to get the address of white pages server 162. White pages server 162 is then accessed to find on which agent host 110 the user's agent 5 (an entity agent) is being hosted. This information is stored in LDAP 140. Client 200 sends the rule to agent host 110, which includes message dispatcher 112. Message dispatcher 112 will route the message to the user's agent 5. The message is parsed by agent 5 as described earlier in the description of the agent architecture. In this case, the placement of the new rule in the expert system would cause the creation of a location sensor within agent 5. This location sensor fires ~~every 30 seconds~~periodically to determine the current location of user 201. The current location may be determined through the use of a stateless task agent, or through a dedicated service running in the society. The task agent or service, in turn, would acquire the location information via an interface offered by the user's ~~cellphone~~cellular telephone service provider.

When the sensor fires, the sensory layer within agent 5 publishes an event that results in the insertion of the current location of the user ~~201~~ within the belief layer of agent 5. This triggers an evaluation of the rules within the reasoning layer of agent 5. A predicate on the left hand side of the rule checks to see if the current location of the user ~~201~~ is within 500 feet of any

know post office. The predicate may make use of a mapping service available on the web to help in making this determination.

If the predicate determines that the user ~~201~~ is within 500 feet of a post office, the rule within the reasoning layer of agent 5 fires. This triggers an action within agent 5, which in turn triggers a single step workflow: sending a message to the user's ~~201's cellphone~~ cellular telephone. The workflow manager 150 picks up this workflow request, and makes use of a stateless task agent to send the text message to the user's ~~201's cellphone~~ cellular telephone.

Many such examples of services could be envisioned besides location based services. These could be things such as, requesting that your agent monitor the price of a stock of a certain company and send signals or messages to the client when the stocks price meets certain criteria or reminding the client of important dates, such as, anniversaries or birthdays, etc.

We have provided a general purpose architecture for both an intelligent agent and a distributed platform for the support of intelligent agents. The invention is not meant to be limited by the scope of the examples used, but is embodied in the claims which follow.

APPENDIX III

(Marked-Up Claims)

4. The architecture of claim 3 wherein said interpreter table contains interpreters that are [is] aware of a domain-specific ontologies and languages [ontology].

12. The architecture of claim 11 wherein said event published by said sensory layer is received by said expert system and said external change is stored in said beliefs storage facility.

39. The society of claim 37 wherein agents are activated by placing an activation entry [them] in a distributed memory area accessible to all entities within said society.

40. The society of claim 39 wherein any one of said plurality of agent hosts within said society may remove said activation entry [an agent] from said distributed memory and run said agent.

41. A society of intelligent agents comprising:

one or more agent hosts for executing a plurality of intelligent agents;

a facilitator for enabling entities external to said society to communicate with and discover information regarding entities within said society; and

a database containing information regarding all agents running in said society;

wherein said intelligent agents comprise:

a messaging facility for handling incoming and outgoing messages;

an expert system for evaluating rules and maintaining known facts;

a sensory facility for sensing conditions external to said intelligent agent; and

a means for communication between said messaging facility, said expert system and said sensory facility.